UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Programación de algoritmos de reducción de ruido en sistemas de adquisición de datos

Raúl Varela Blázquez
Tutor: Fernando Jesús López Colino
Ponente: Guillermo González de Rivera Peces

JUNIO 2017

PROGRAMACIÓN DE ALGORITMOS DE REDUCCIÓN DE RUIDO EN SISTEMAS DE ADQUISICIÓN DE DATOS

AUTOR: RAÚL VARELA BLÁZQUEZ
TUTOR: FERNANDO JESÚS LÓPEZ COLINO

Hardware & Control Technology Laboratory
Escuela Politécnica Superior
Universidad Autónoma de Madrid
junio de 2017

Resumen (castellano)

En este Trabajo Fin de Grado se desarrollará un software para reducir el ruido en la adquisición datos de un sistema de medida mediante un sistema embebido. Utilizaremos el procesador de tiempo real (PRU) de la placa BeagleBone Black para liberar el procesador principal de esta placa para su uso en otras funciones.

El primer objetivo consiste en la utilización de un compilador de dicho sistema para que, a partir de un código en C, se genere un código ensamblador específico de la propia PRU para el tratamiento de los datos transmitidos por el sistema de medida.

Una vez realizado esto, se procede a analizar diferentes algoritmos de reducción de ruido para comprobar su efectividad y utilidad para la función que queremos desempeñar. Determinados cuales son los algoritmos que funcionan con mayor eficiencia, se procederá a adaptarlos para mejorar la limpieza de la señal.

Como resultado de este proyecto la calidad e integridad de los datos proporcionados por el sistema de medida mejorarán significativamente mediante este pre procesado.

Abstract (English)

The objective of this End of Degree Thesis is to document the development of a software that will reduce noise in the process of data acquisition for a measuring system by means of an embedded system. The workload will be handled by the real-time processor of a BeagleBone Black board, this is to take advantage of processing data in real time and also to alleviate the workload of the main processor.

The first objective is to a compiler of the entire system in order to obtain an assembler code from the original code in C. The generated code will be eligible to run in the PRU in order to treat the data transmitted by the measuring system.

Once this has been completed we will proceed to analyze different noise reduction algorithms in order to verify their effectiveness and utility for the desired case scenario. When the best algorithms have been determined we will proceed to adapt them in order to obtain a cleaner signal.

As a result of the outcome generated by this End of Degree Thesis, the quality and integrity of the data provided by the measuring system will be significantly improved as a result of the applied algorithms.

Palabras clave (castellano)

BeagleBone, sistema de medida, PRU, tiempo real, algoritmos, ruido.

Keywords (inglés)

BeagleBone, measuring system, PRU, real time, algorithms, noise.

Agradecimientos

En primer lugar, dar las gracias a todos los profesores que nos han hecho mucho más agradable el paso por esta complicada parte de nuestra vida, invirtiendo su tiempo en que aprendamos de ellos todo lo que podamos. Quería en especial dar las gracias al HCTLAB y a mi tutor, Fernando por darme la oportunidad de trabajar con ellos y aprender grandes cosas desarrollando este proyecto.

En segundo lugar, dar las gracias a todas esas personas, sin las que no habría sido posible sacarse la carrera gracias a sus ánimos y apoyo continuos. En especial a Sergio, Víctor, Garay, Ricardo, Gonzalo y todas esas personas que han dado la cara por mi todo este tiempo.

Por último, pero no menos importante, a mi familia, que ha sacarificado tantas cosas para que llegue donde he llegado, por ayudarme a superar todas las barreras que en ocasiones me ponía. Miles de gracias a mi madre, mi abuela y mi madrina por su comprensión, por su insistencia, y por estar ahí para mí día tras día. En especial, dar las gracias a mi hermana pequeña, Elena, que ilumin todos los días de mi vida y a mi padre. Esto va por él.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

1.1 Motivación

En la actualidad, el empleo de sistemas embebidos ha aumentado notablemente debido a su facilidad de uso, su pequeño tamaño y sus grandes posibilidades a la hora de desempeñar funciones específicas. Esta polivalencia junto con el bajo coste y las grandes prestaciones que poseen algunos de estos sistemas embebidos hacen que desarrollen de manera muy efectiva la función para la que son requeridos. La motivación de este trabajo es la de poder incluir la reducción de ruido en un procesador auxiliar de un sistema embebido.

1.2 Objetivos

El objetivo principal es el de implementar y evaluar algoritmos un algoritmo que elimine o atenúe de la forma más eficiente posible el ruido que presenta un sistema de adquisición de datos.
El objetivo secundario es usar procesadores de tiempo real de una BeagleBone[8] para realizar el proceso de adquisición y adaptación de datos. Para habilitar el desarrollo se busca trabajar con los compiladores específicos de la PRU[14].
Para lograrlo, se desarrollarán en lenguaje C dichos algoritmos de manera que se usen las PRU de nuestro sistema embebido para que trabaje a tiempo real, dejando libre el procesador principal para otras gestiones futuras de la placa tratando de evitar usar ensamblador para su facilidad tanto a la hora de implementarlo como a la hora de optimizarlo y mejorarlo en un futuro.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- En el primer capítulo se habla sobre los objetivos generales de este proyecto, se da una breve introducción sobre los recursos que vamos a utilizar, la forma en la que se va a realizar y la motivación por la que se realiza dicho proyecto de desarrollo.

- En el segundo capítulo, el Estado del Arte, se hará una primera introducción a los sistemas embebidos y sus ventajas actuales, se hablará también de los microprocesadores, las ventajas de trabajar a tiempo real y por ultimo de algoritmos de reducción de ruido.

- En el tercer capítulo se hablará de todo lo que ha sido el desarrollo del proyecto, desde las primeras pruebas para poder compilar un código en C para trabajar con las PRU a tiempo real hasta el desarrollo del algoritmo para la reducción de ruido de nuestro sistema de adquisición de datos.

- En el cuarto capítulo se detallarán las pruebas realizadas, los algoritmos de reducción de ruido implementados y la comparación final de dichos algoritmos.
- Finalmente se hará una breve conclusión y posibles mejoras futuras de todo lo realizado.

2 Estado del arte

En este apartado se explican las bases sobre las que se ha desarrollado este Trabajo de Fin de Grado, partiendo de las diferentes partes del sistema de adquisición de datos, dispositivos utilizados para la adquisición medida y las diferentes técnicas de tratamiento de la señal utilizadas para minimizar el ruido de forma que se obtengan unos resultados lo más preciso posible.

2.1 Sistema de Adquisición de Datos

La adquisición de datos consiste en la obtención de muestras de una señal real en formato digital que puedan ser manipulados por un ordenador o un sistema de control [2] .
Un sistema sensor está compuesto por un transductor con electrónica de acondicionamiento y un convertidor analógico-digital.

2.1.1 Elemento transductor

Un sensor es un dispositivo que utiliza sus capacidades físicas con el fin de adaptar la señal registrando un cambio de magnitud física, química o biológica para convertirlo en una señal eléctrica que pueda ser medida [3].
Dependiendo de la magnitud que se desea medir existen multitud de tipos de sensores como son los de humedad, presión y temperatura que suelen ser realizados mediante componentes pasivos como resistencias variables, capacidades eléctricas o termopares y componentes activos.
Si se estudia la estructura de un transductor está formado por el elemento que se encuentra en contacto con la magnitud física a medir y una parte de transducción que transforma esa señal analógica del sensor en un valor interpretable eléctricamente. Los principales tipos de transductor son los siguientes [4] :

- Piroeléctrico: Genera una carga eléctrica al ser expuesto a calor en forma de radiación infrarroja.
- Electroquímico: Genera una corriente o voltaje a partir de los parámetros químicos de las sustancias a medir.
- Resistivo: Variación de la resistencia eléctrica.
- Capacitivo: Genera un aumento o disminución de la capacidad del condensador.

- Piezoeléctricos: Genera un campo eléctrico al desplazarse las cargas tras aplicar una tensión mecánica sobre el material.
- Inductivo: Se genera un voltaje a través del movimiento de la espira conductora interior.

2.1.2 Dispositivo electrónico de Adquisición de Datos (DAQ)

Este dispositivo se encarga de la interpretación del voltaje o corriente obtenida del transductor explicado en el apartado anterior. Para interpretar los diferentes datos se debe conocer con exactitud el tipo de señal que va a ser recibida junto con los siguientes datos:

- Rango de valores. Es necesario conocer estas características para adaptar el conjunto de valores que van a ser interpretados y no encontrar situaciones de saturación o de resolución de baja calidad.
- Resolución necesaria. Dependiendo del tipo de medida que se vaya a realizar es necesario conocer el número de valores que son necesarios a nivel de precisión.
- Sistema o dispositivo que se va a utilizar para interpretar el conjunto de medidas.
- Velocidad de muestreo.

A partir de estos valores se construye el Hardware necesario para la conversión de los valores mediante la elección de los dispositivos que forman parte de un circuito impreso ([Ilustración 1]).

Ilustración 1: Sistema de Adquisición de Datos [4]

2.2 Sistemas embebidos

Durante los últimos años han aparecido en el mercado dispositivos digitales llamados sistemas embebidos que surgieron en un principio para labores sencillas de control. Estos sistemas han evolucionado de forma que se han desarrollado como pequeños ordenadores

que se pueden adquirir por un coste mínimo y que incluyen algunas características de gran utilidad como conexiones wifi, puertos HDMI o incorporando pequeñas pantallas con una gran resolución [5] .

Aunque también existen sistemas embebidos sin sistema operativo para la realización de funciones más simples, la introducción en los mismos de un sistema operativo nos añade una amplia gama de posibilidades a la hora de utilizarlos. Estos dispositivos son capaces de comunicarse con dispositivos externos usando su sistema operativo embebido que incorpora una gran cantidad de funciones heredadas de los Sistemas Operativos de propósito general que actualmente se encuentran en nuestros portátiles u ordenadores de sobremesa ([Ilustración 2]).

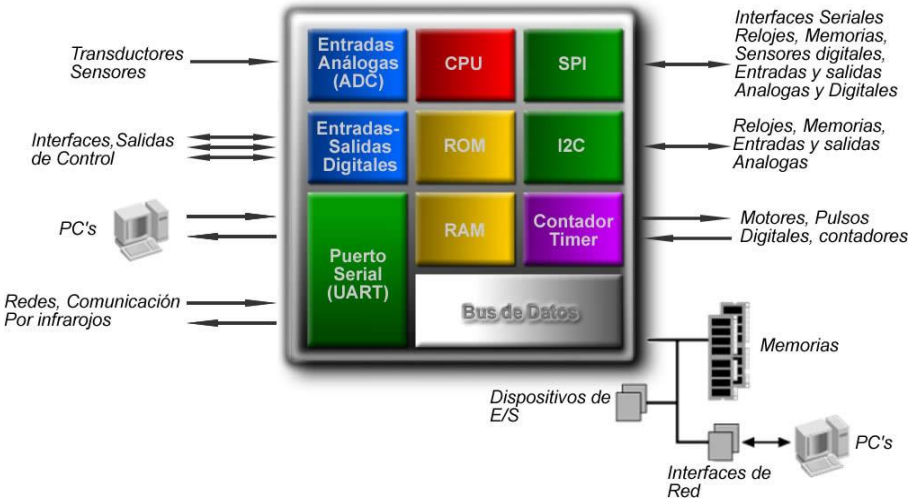


Ilustración 2: Esquema general de un sistema embebido [6]

En este proyecto ha sido utilizado el sistema BeagleBone Black[8]. Esto es debido a las características de su procesador, las cuales se estudiarán en profundidad en el siguiente punto, junto con sus unidades programables en tiempo real y la capacidad de incorporar nuevos dispositivos o funcionalidades a partir de su conjunto de conexiones.

2.2.1 BeagleBone Black

La BeagleBone Black es una placa de Hardware libre producido por Texas Instrument y cuya fabricación la llevan a cabo las compañías DigiKey y element14 y aunque fue ideada como un dispositivo educacional[15] ha adquirido características profesionales. Su procesador es el AM3358 Cortex A8 ([Ilustración 3]) de 32 bits cuya velocidad viene determinada por la alimentación del dispositivo, por tanto es de 500MHz conectado vía USB o 750MHz conectado con una fuente DC.

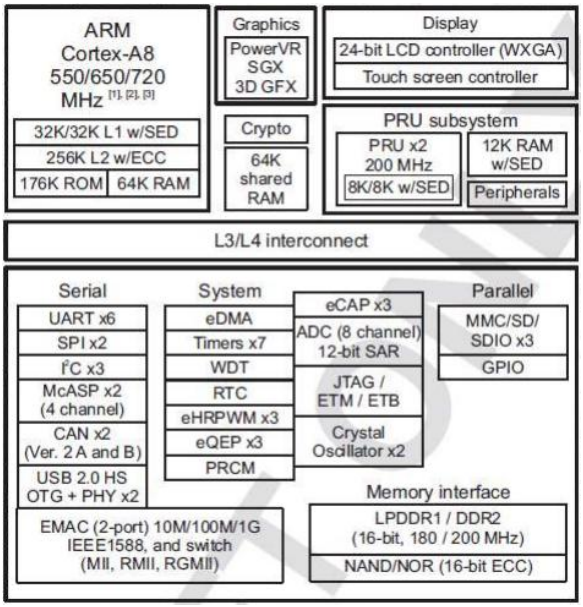


Ilustración 3: Procesador AM3358 [7]

La característica más relevante para este Trabajo de Fin de Grado de este procesador es que incorpora dos unidades programables a tiempo real llamadas PRU. Éstas consisten en dos módulos independientes al procesador que permiten realizar tareas paralelamente sin suponer un coste computacional añadido. Estas unidades se comunican con el procesador principal mediante el módulo de comunicación PRU-ICSS y comparten información mediante la memoria RAM.

Para la conexión de dispositivos externos, el dispositivo BeagleBone dispone de conexiones Ethernet, USB y dos arrays de 46 pines paralelos que se pueden observar en la [Ilustración 4].

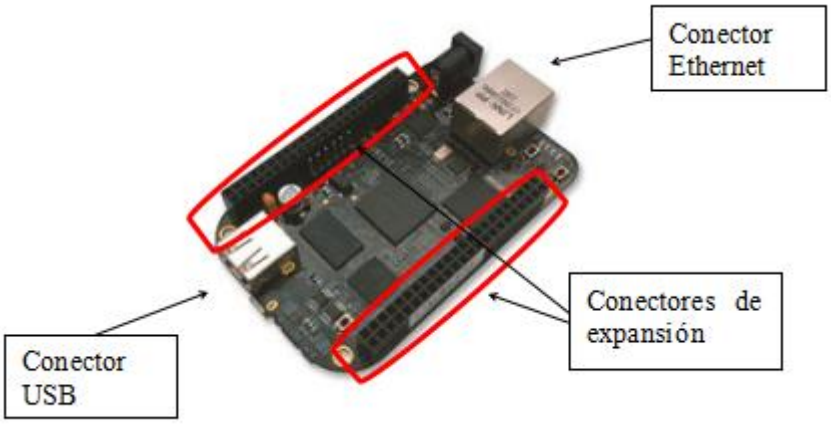


Ilustración 4: BeagleBone Black

Algunas de las capas de extensión más comunes para este dispositivo son las que incorporan elementos Hardware como el conector HDMI que permite la conexión a un monitor o una pantalla táctil adicional como las que se muestran en las siguientes imágenes y que permiten utilizar el dispositivo como un ordenador o Tablet [8]

2.3 Ruido eléctrico

Se conoce como ruido eléctrico ([Ilustración 5]) todas aquellas perturbaciones que producen interferencias no deseadas en la señal principal. Estas perturbaciones son causadas por el ruido intrínseco de los componentes, sistemas de fabricación humana y perturbaciones causadas por agentes naturales en el dispositivo [9] .

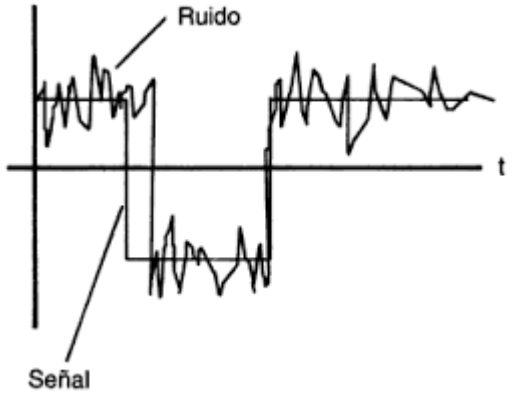


Ilustración 5: Señal con ruido [10]

Para minimizar el ruido de los circuitos se aplican las siguientes los siguientes criterios durante el proceso de diseño del Hardware:

- Evitar grandes giros en las pistas.
- Elegir componentes de bajo ruido.
- Realizar un plano de masa digital y otro plano analógico unidos mediante un material resistivo.
- Utilizar pistas lo más corto posibles con el grosor y separación adecuada.
- Separación del voltaje digital y analógico.
- Utilización de condensadores próximos a la alimentación del componente para eliminar los picos de voltaje.
- Uso de condensadores para eliminar la oscilación del sistema.

Dado que es imposible eliminar el ruido por completo de los componentes o del circuito, se utilizan diferentes algoritmos o técnicas para eliminar o suavizar el mismo del conjunto de medidas.

2.4 Algoritmos de reducción de ruido

Existen multitud de algoritmos y técnicas para eliminar ruido de los resultados digitales obtenidos de las medidas, desde las diferentes técnicas de muestro o filtrado inicial de la señal, hasta el uso de algoritmos promedio o estadísticos [16]. Los principales están basados en la media de los valores anteriores y siguientes de la muestra y métodos de comparación con señales anteriores o patrones.

2.4.1 Métodos basados en la media

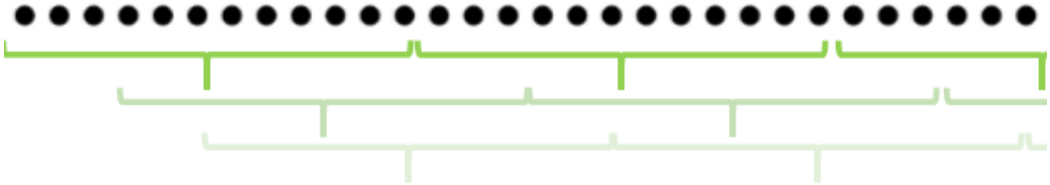
Lo primero a tener en cuenta a la hora de aplicar estos algoritmos es en la frecuencia de la señal y del ruido de la misma. La frecuencia del ruido es la que nos va a hacer decidir el número de muestras a mediar, que aumentará a mayor frecuencia del ruido. Por otro lado también hay que tener en cuenta la frecuencia a la que funciona la señal, ya que a la hora de decidir cuantas muestras mediar, la frecuencia a la que trabaje la señal es la que nos va a poner dicho límite ya que si se producen cambios en la misma a una frecuencia relativamente baja, no interesa mediar valores que son iguales. En resumen, al aplicar estos algoritmos tendremos que ajustar la frecuencia de muestreo a la de la señal y el número de muestras a mediar a la frecuencia de ruido introducido en la señal objetivo.
Para la obtención del valor más preciso posible se recogen arrays de valores cuya longitud depende del porcentaje de ruido o número de perturbaciones que se producen en el canal. A partir de esos valores, se aplican algoritmos que detectan si uno de los valores es inusual o incorrecto ya que se produce una desviación porcentual en la media de valores.
El valor incorrecto se corrige mediante la aplicación de fórmulas matemáticas sobre el conjunto de valores para estimar el valor que debería ser correcto. Un punto de gran importancia en este tipo de algoritmos es la selección de los bloques de muestreo, ya que la misma muestra debe encontrarse en varios bloques para evitar la estimación incorrecta en las muestras que inician o finalizan el bloque.
En las siguientes imágenes se muestra cómo realizar el muestreo y los problemas explicados anteriormente.

Supongamos esta secuencia de muestras

Inicialmente hacemos un muestreo simple, tomando conjuntos de 12 muestras:

Suponiendo un error en la segunda muestra del bloque, y debido a que sólo mediamos una vez dicho bloque, el error será significativo.

Si en vez de pasar directamente al mediar el siguiente bloque, reutilizamos valores del bloque anterior para volver a realizar la media, el resultado global de estas medidas se acercará mucho más al valor sin error que esperamos conseguir



Como se puede observar, un menor espaciado entre los bloques, entendiendo este concepto como el número de muestras que no vamos a reutilizar entre bloque y bloque, puede aumentar el número de veces que se repite la muestra con error, sin embargo al tener un amplio conjunto de bloques mediados, el resultado final será mucho más cercano a la medida real objetivo.

Una vez planteado como realizar el proceso de muestreo y las principales características de estos métodos, se explican las diferentes fórmulas que se pueden aplicar [11].

Media aritmética

Este método consiste en la suma de los valores anteriores y siguientes y la división entre el número de muestras.

$$\tilde{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + x_3 + \ldots + x_n}{n}$$

Este sistema de medida puede verse afectado en distribuciones muy asimétricas y es afectada por factores externos.

Media geométrica

Consiste en la obtención del resultado de la raíz enésima del producto de todas las muestras.

$$\tilde{x} = \sqrt[n]{\prod_{i=1}^{n} x_i} = \sqrt[n]{x_1 \cdot x_2 \cdot x_3 \ldots \cdot x_i}$$

Los resultados se ven influenciados en el caso de existir valores pequeños.

Media armónica:

El resultado es el recíproco de la suma de los recíprocos multiplicado por el número de elementos del conjunto.

$$\tilde{x} = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}} = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + + \frac{1}{x_n}}$$

Este método puede inducir problemas ya que no permite el valor 0.

Media cuadrática:

Este método consiste en el resultado de la raíz de la suma de los cuadrados de los valores y su división por el número de muestras.

$$\tilde{x} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2} = \sqrt{\frac{x_1^2 + x_2^2 + x_3^2 + ... + x_n^2}{n}}$$

2.4.2 Métodos basados en comparación

Las técnicas basadas en comparación son más complejas que las vistas anteriormente y que utilizaban los diferentes tipos de media. Si las medidas que se van a realizar muestran siempre el mismo patrón se pueden realizar diversos algoritmos que permitan adaptar los valores perdidos en la prueba actual. Para ello y a partir de un conjunto de medidas que puedan ser utilizadas como entrenamiento, se pueden crear diversos algoritmos monitorizando el tiempo de la muestra y calculando la fórmula que genera el elemento a medir[17]. Los más destacados se citan a continuación.

- Árboles de decisiones
- Reglas de asociación
- Redes neuronales
- Maquinas de vectores
- Algoritmos de agrupamiento
- Redes bayesianas

Debido a la complejidad y eficacia de estos métodos, resulta inviable realizarlos en este proyecto. Sin embargo, la metodología vectorial y de agrupación van a servir para enfocar la implementación algoritmos más efectivos en este proyecto.

3 Diseño y desarrollo

3.1 Introducción

En este apartado del Trabajo de fin de Grado se va a explicar todo el desarrollo y la operativa llevada a cabo para la realización del proyecto. Ahondando en cómo obtenemos los datos, los elementos que vamos a usar para desarrollar el sistema completo, características de dichos elementos y, por último, cómo se ha llevado a cabo la ejecución de este proyecto.

3.2 Sistema de medida

3.2.1 ADS1146

Para la realización de las pruebas, vamos a utilizar el conversor analógico-digital (ADC) ADS1146 conectado en una placa a un potenciómetro. Este ADC se configurará para que trabaje a una frecuencia de 2000 muestras por segundo (que será también la frecuencia de muestreo en la PRU), y cuyas lecturas vendrán dadas por 16 bits de información. El funcionamiento de esta placa viene descrito a continuación.

Mediante una alimentación externa (utilizaremos la propia de la BBB de 3,3 V) y el potenciómetro de la placa, controlaremos la alimentación que le va a llegar al ADS1146 (que en este caso irá de 0V a 3,3V). Esto nos servirá para la configuración y pruebas iniciales de medida de la BBB.

La comunicación entre el ADC y la BeagleBone se realizara mediante SPI. En un principio el ruido que introduce el sistema es mínimo. Posteriormente se le añadirá una fuente de ruido para comprobar el funcionamiento de los algoritmos.

Una vez configurada la comunicación, se sustituyó la salida del potenciómetro por una señal ruidosa generada por el osciloscopio MSO-X 3104A y se procedió a realizar las pruebas de los algoritmos implementados.

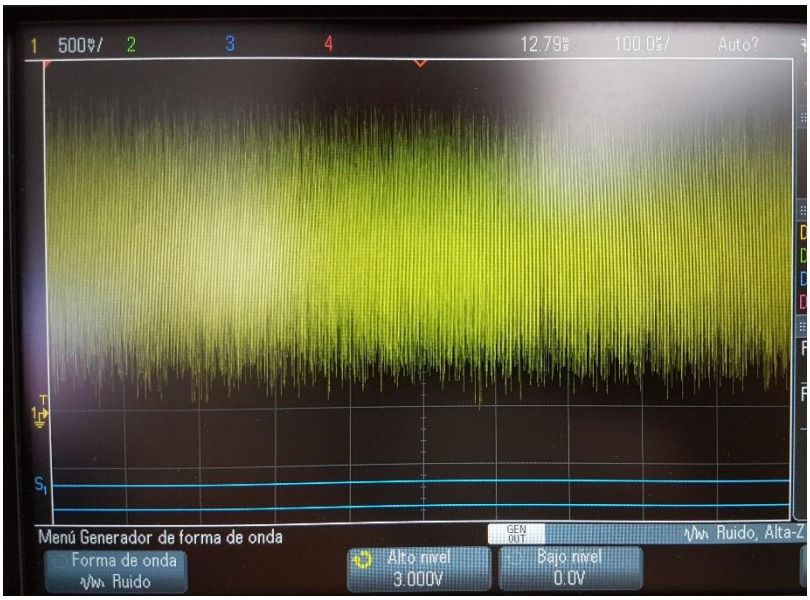


Ilustración 6: Señal con ruido amplitud 1,5V

Esta señal ruidosa ([Ilustración 6]) tiene una amplitud tan grande para compensar que el propio ADC tiene su propio filtro de ruido. En este TFG interesa conservar la mayor parte posible de dicho ruido.

La configuración de esta ADC se realiza mediante la transmisión de 8 bits (por SPI) con la instrucción que deberá ejecutar el mismo. De esta forma se dejará configurado para que solo lea datos cuando se lo solicitemos, su ganancia sea uno y trabaje a la máxima frecuencia que admita dicho componente [12].

3.3 BeagleBone Black

3.3.1 Configuración inicial

Una de las características más importantes de la BBB es el poder mapear sus I/O de la forma que queramos mediante “Device Tree Overlay”, gracias al cual podemos indicar, además, cómo queremos que se comporten. En nuestro caso, vamos a utilizar los pines de forma que podamos establecer una comunicación SPI de las PRU con nuestro aparato de medida [13]

El mapeo de dichos puertos estará realizado en .dts que será compilado y cargado en la BBB para trabajar con dichos puertos configurándolos como salidas o entradas de esta manera:

0x1a4 0x0d // CS   P9_27 pr1_pru0_pru_r30_5, MODE5 | OUTPUT |
0x19c 0x2e // MISO P9_28 pr1_pru0_pru_r31_3, MODE6 | INPUT |
0x194 0x0d // MOSI P9_29 pr1_pru0_pru_r30_1, MODE5 | OUTPUT |
0x198 0x0d // CLK P9_30 pr1_pru0_pru_r30_2, MODE5 | OUTPUT |

Estos archivos están contenidos en el directorio \overlay dentro del directorio principal del proyecto.
Adicionalmente se configuró también una de las I/O para poder generar interrupciones directamente en la PRU. Estas interrupciones están configuradas con el fin de poder parar el proceso que esté llevando a cabo manualmente, no sólo con la finalización del código que se esté ejecutando en la misma.

0x190 0x26 // INTER P9_31 pr1_pru0_pru_r31_3, MODE6 | INPUT |

3.3.2 Memoria compartida

En la BBB, las PRU comparten memoria con el procesador principal de la siguiente manera [Ilustración 7].

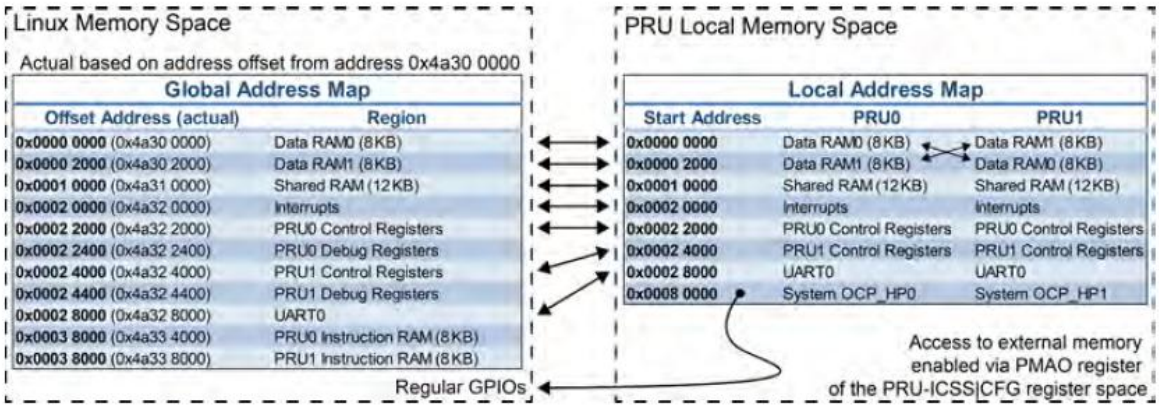


Ilustración 7: PRU-ICSS Mapa de direcciones de memoria [13]

Para acceder a esta memoria, (que contendrá la información de las muestras que va a medir nuestro ADC y a comunicar a la PRU de la BBB) se va a utilizar una herramienta llamada mem2file (diseñada por Derek Molloy [13]) cuya función es la de representar las muestras medidas.
Uno de los problemas, a la hora de trabajar con memoria, consiste en que las PRU, en su memoria Ram, tiene elementos ya ocupados, con lo que no podremos usar dicha memoria para guardar datos, ya que corremos el riesgo de que no se guarden bien, o que se alteren los resultados antes de extraerlos. Es por esto que nuestro programa principal va a mandar a las PRU, por parámetro, una dirección de memoria específica, solventando este problema, dejando libre la memoria propia de la PRU y usando piscinas de datos propias del procesador principal, para así poder representar de la forma más amplia posible todas las pruebas que vamos a realizar.

3.3.3 Ejecución

Para la realización del proyecto, se va a proceder a usar la PRU0 (Unidad Programable en tiempo Real) del procesador de la BBB (Sitara XAM3358BZCZ100 Processor). Esto es debido a la necesidad de dejar libre el procesador principal para la realización de otras tareas simultaneas al de las PRU.

Uno de los grandes problemas que suponía trabajar directamente en ensamblador era la dificultad de realizar operaciones más complejas, necesarias para la implementación de los algoritmos, en dicho lenguaje. Por este motivo, el planteamiento del TFG es del de utilizar las herramientas disponibles para permitir el uso de lenguaje C.

Se optó por usar la herramienta proporcionada por Texas Instruments "TI PRU C Compiler" [14] que nos va a permitir trabajar con las PRU utilizando el lenguaje de programación C pudiendo intercalar directamente código ensamblador para poder llevar a cabo la comunicación SPI y el trabajo con memoria, lo cual es estrictamente necesario para poder extraer los datos medidos por el ADC desde las PRU hasta el procesador principal.

El resultado de la compilación (comandos de compilación en anexo [A]), será mandado a las PRU por un programa principal (testBBB.c) recogido en el anexo [B] que, además de inicializar y configurar las PRU, les pasara por parámetro la posición de memoria donde guardar las muestras. El límite de muestras a guardar y los bits de comunicación que transmitir por SPI entre otras cosas.

En todos los casos, todos los algoritmos constaran de un testPRU.c, que contendrá el "main" principal del proyecto, se implementaran los algoritmos y donde se llamará a las funciones contenidas en testHAL.c (anexo [C]) para una mayor limpieza del proyecto y además de usar las funcionalidades del compilador anteriormente citado, que nos permite poder trabajar con variables directamente en ensamblador. Esta característica es la que nos permite combinar el funcionamiento de los GPIOs de la BBB y la implementación de los algoritmos con la respuesta de los mismos directamente en lenguaje C pudiendo usar algoritmos más complejos de una forma mucho más visual y directa que si trabajáramos solamente en lenguaje en ensamblador.

Inicialmente se realizó un programa para verificar el correcto funcionamiento del diseño sin aplicar ningún tipo de algoritmo. Este consiste en realizar la comunicación SPI y guardar las muestras en memoria para su posterior representación (\multiPussCSpi) anexo [D].

Posteriormente, se procedió a la implementación de algoritmos de reducción de ruido. Estos son:

- Media aritmética (\mediaAritmetica) anexo [E]
- Media armónica (\mediaArmonica) anexo [F]
- Media cuadrática (\mediaCuadratica) anexo [G]
- Media aritmética adaptativa (\aritmeticaAdaptativa) anexo [H]

- Media armónica adaptativa (\armonicoAdaptativo) anexo [I]

3.3.4 Limitaciones técnicas en la compilación del programa

La limitación más importante, a la hora de compilar, es la de disponer solamente 8kB ([Ilustración 8]) para la paginación de instrucciones en las propias PRU, lo que ha impedido realizar programas de una complejidad excesiva y utilizar funciones predefinidas para la realización de operaciones más complejas (elevar, raíz cuadrada, raíz enésima, ...). Por estos motivos, cada algoritmo tiene su propio programa, no dejando la posibilidad de poder tenerlos todos en un mismo código y permitiendo que mediante el cambio de un simple parámetro podamos elegir el algoritmo que deseamos usar.

```
PAGE 0:
  PRUIMEM:    o = 0x00000000  l = 0x00001000  /* 8kB PRU0 Instruction RAM */
PAGE 1:
  PRUDMEM:    o = 0x00000000  l = 0x00001000  /* 8kB PRU Data RAM 0 */
```

Ilustración 8: Paginación de la PRU

4 Integración, pruebas y resultados

4.1 Pruebas y medidas iniciales sin ruido

En un principio, para comprobar si el sistema se ha configurado correctamente iniciamos una tanda de medidas con un ADC con un ruido mínimo y entre medias conectamos un osciloscopio para comprobar que se está realizando la comunicación SPI entre las PRU y nuestro ADC.

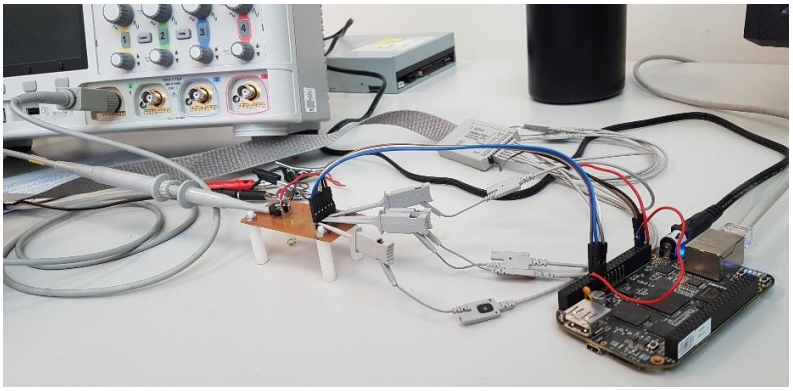


Ilustración 9: Conexión del sistema

Tras mandar los bits de configuración al ADC, este comienza a convertir el voltaje, que le proporciona el potenciómetro, a digital y a mandárselo a la PRU. Durante la comunicación SPI ([Ilustración 10] y [Ilustración 11]), la BBB va a transmitir 24 bits, 8 de los cuales (0x12) indican al ADC que comience una lectura. Durante los otros 16 bits el ADC va a responder con la lectura del valor convertido.

Ilustración 10: Comunicación SPI

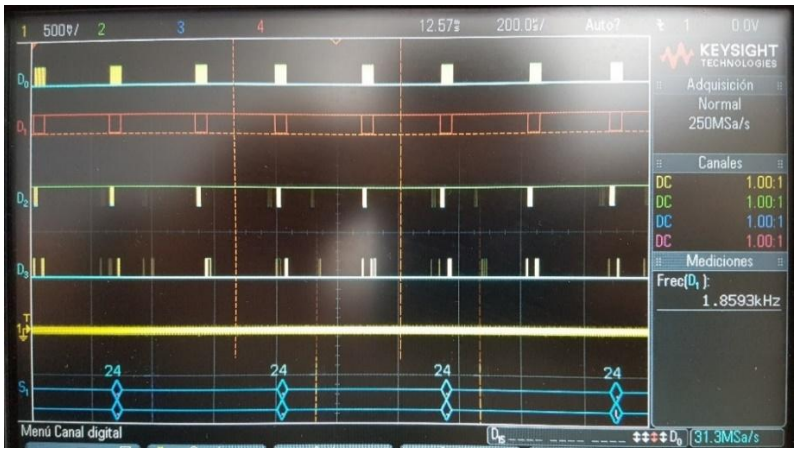


Ilustración 11: Comunicación SPI y frecuencias de funcionamiento

Inicialmente se realiza la medida de dos señales sin ruido, una señal estable y otra señal que varía a determinada frecuencia, con el objetivo de comprobar el ruido propio del diseño y la frecuencia que es capaz de soportar.

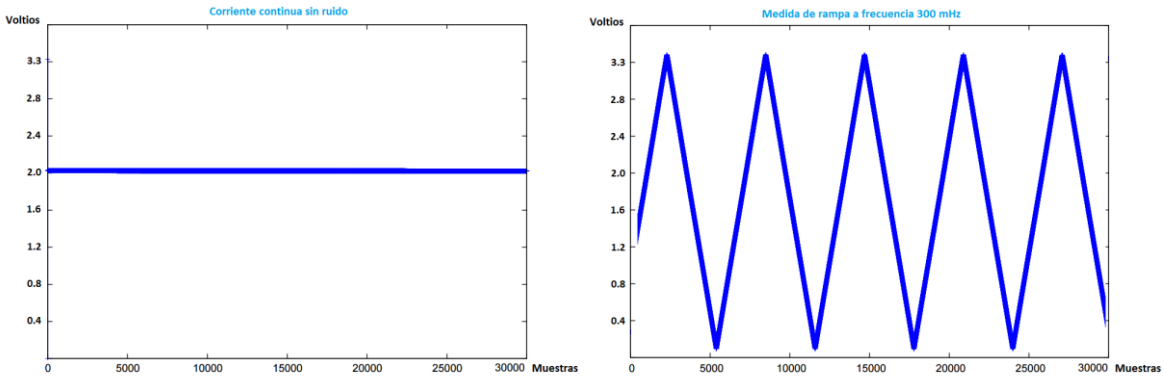


Ilustración 12: Señales y medidas sin ruido

Como se puede observar ([Ilustración 12]), el ruido propio del sistema es inferior a 0,1V midiendo con exactitud las dos señales anteriores y nuestro sistema funciona correctamente a una frecuencia de 2000Hz, siendo esta una limitación propia del ADC, ya que la BBB es capaz de funcionar con una frecuencia mayor.

4.2 Pruebas y medidas finales con ruido

Una vez comprobadas que las medidas eran correctas, se procedió a la introducción de ruido y a su posterior medida. Este ruido se genera mediante el osciloscopio anteriormente citado, usando una sonda que irá directamente conectada a la entrada del potenciómetro, de forma que podamos controlar la salida del mismo, ya que este voltaje de salida es el que medirá el ADC.

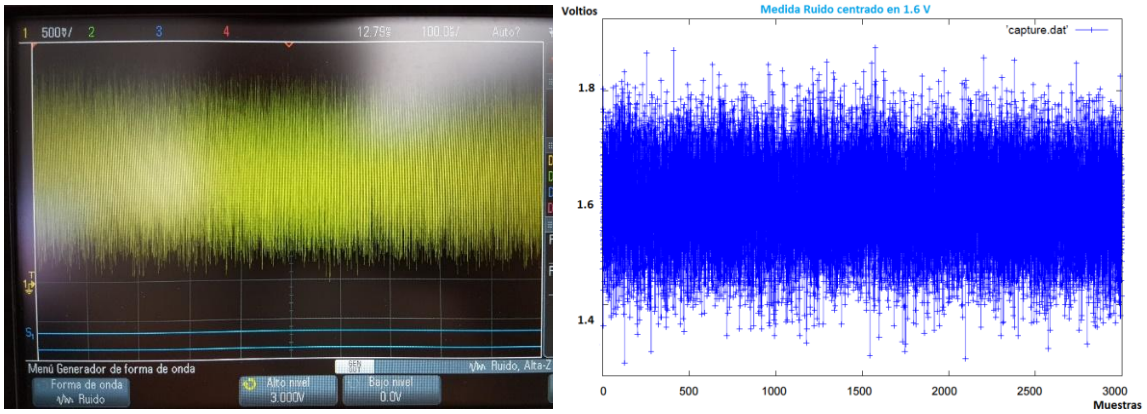
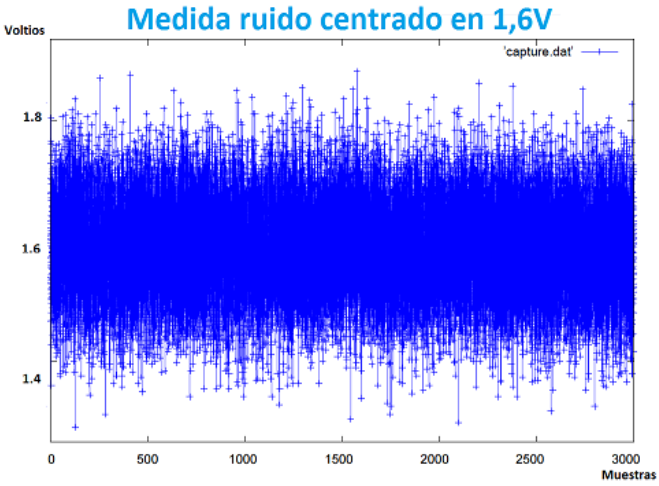
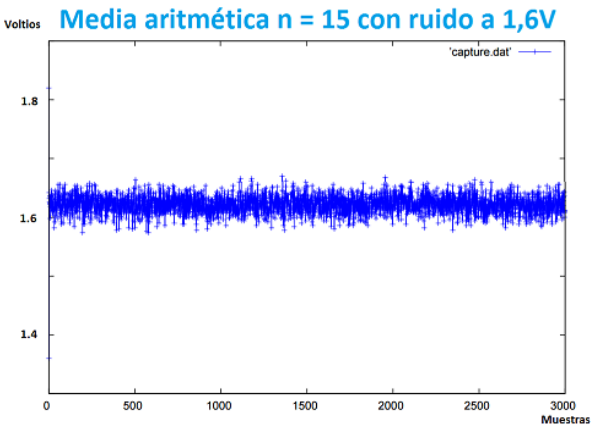


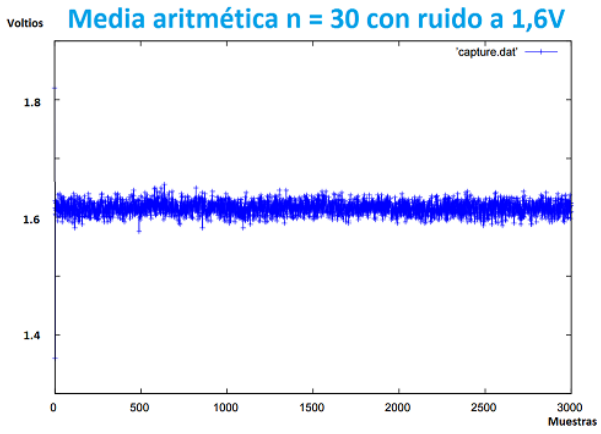
Ilustración 13: Señal ruidosa

Como se puede observar ([Ilustración 13]), de la señal original de 1,5V de amplitud del ruido, el propio ADC limpia dicho ruido, dejando un ruido de apenas 0,4V sobre el que se realizará la prueba de los algoritmos de reducción.

- Algoritmo 1: Media Aritmética







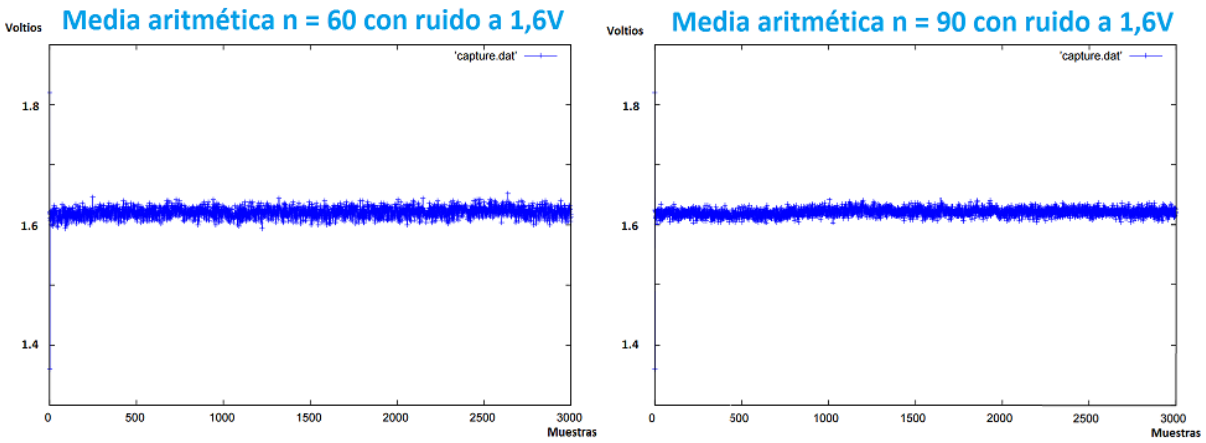
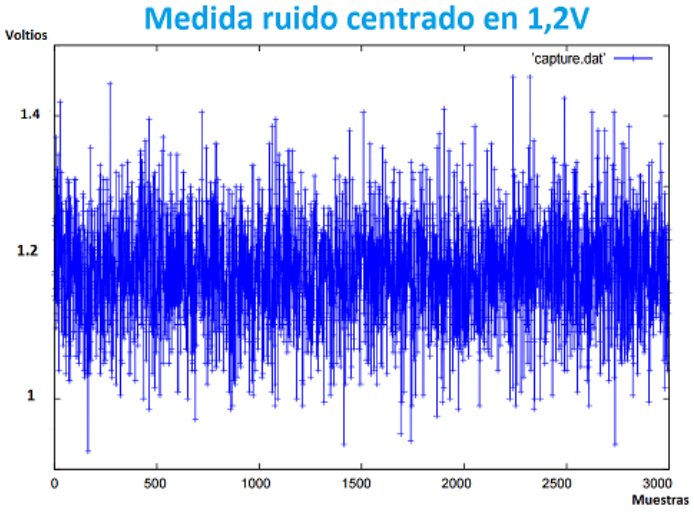
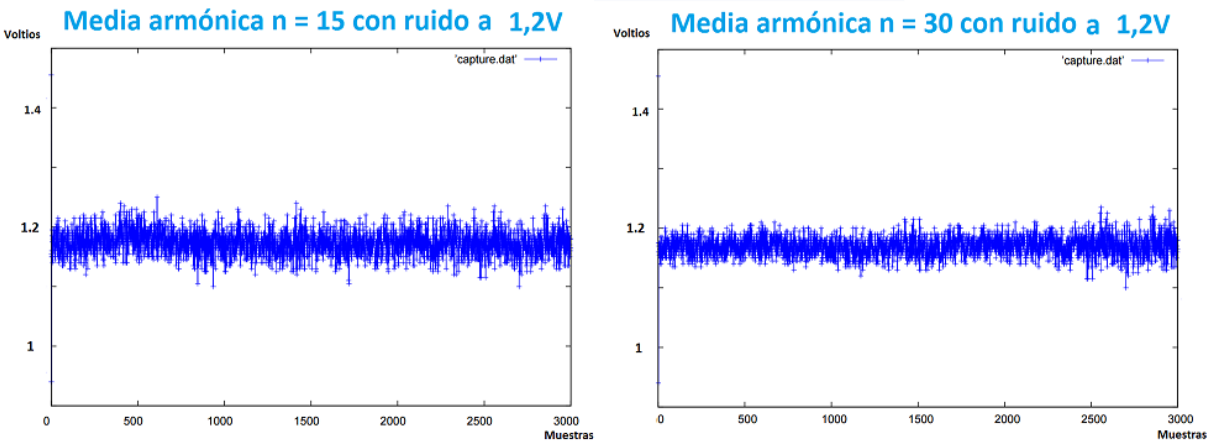


Ilustración 14: Media aritmética

- Algoritmo 2: Media Armónica





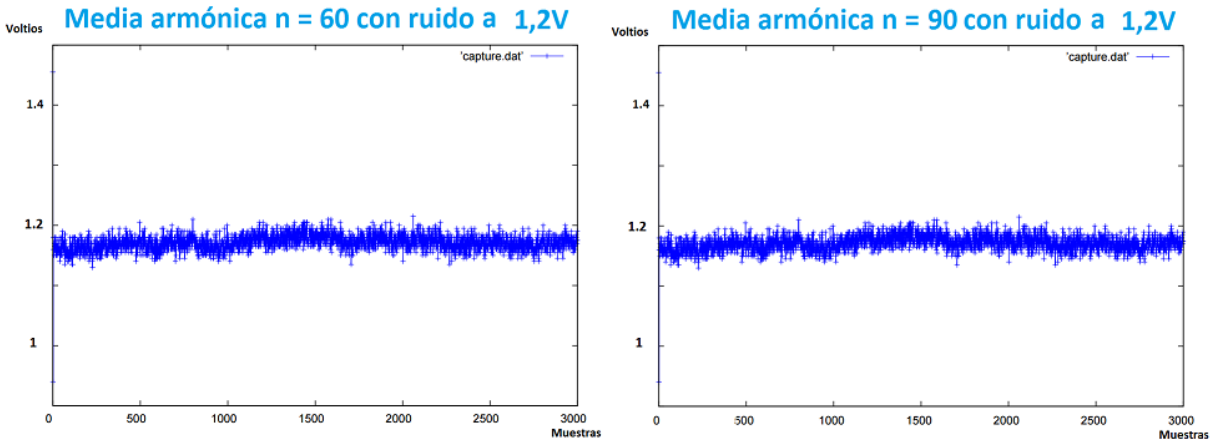
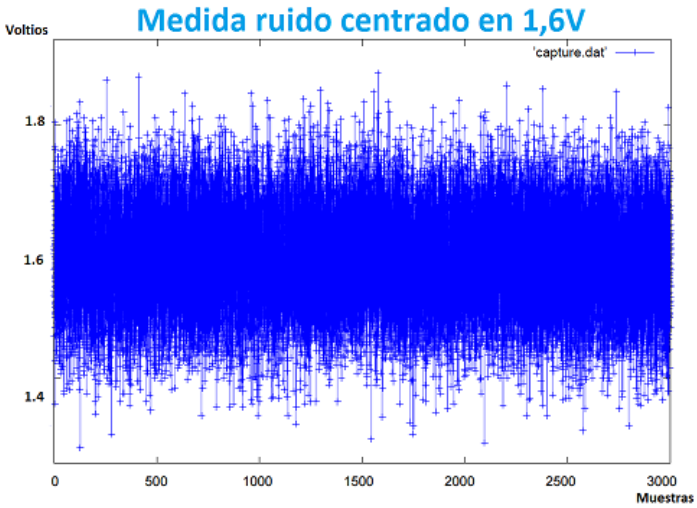
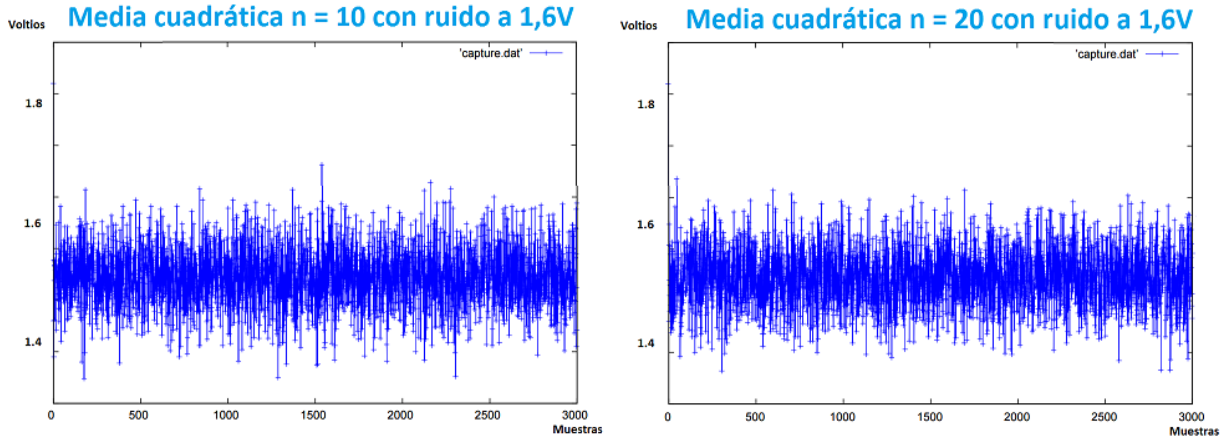


Ilustración 15: Media armónica

- Algoritmo 3 : Media Cuadrática





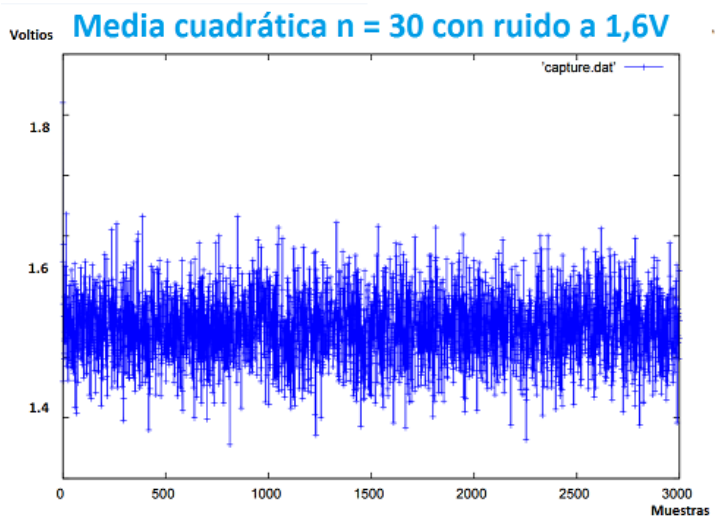


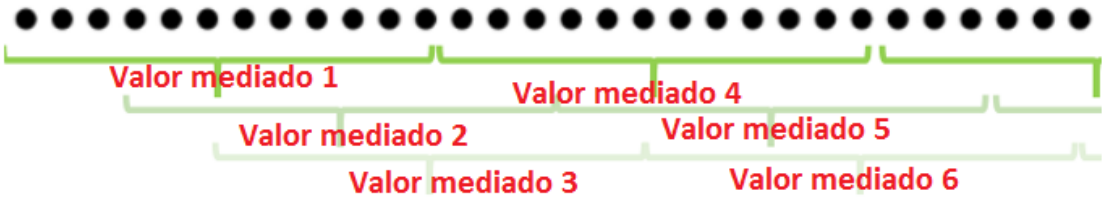
Ilustración 16: Media cuadrática

En vista de los resultados obtenidos, se puede observar claramente que la media aritmética ([Ilustración 14]) es la que ofrece una limpieza de ruido superior, seguida muy de cerca por la media armónica. La media aritmética requiere menos coste computacional y ofrece la posibilidad de mediar un numero enorme de muestras sin que se vea gravemente afectado el rendimiento del sistema, pudiendo trabajar a mayor frecuencia debido al escaso número de operaciones que hay que llevar a cabo para realizarla. La media armónica, por el contrario, tiene un coste computacional mucho mayor debido al gran número de divisiones que se han de realizar y la señal no mejora en gran medida al aumentar el número de muestras a mediar ([Ilustración 15]). Sin embargo, la eficacia de los dos algoritmos anteriores es muy superior al de la media cuadrática ([Ilustración 16]), cuyos resultados son claramente peores, sin apenas mejorar la calidad de la señal mediada, debido en gran medida a la propia limitación de la función implementada para la realización de las raíces, siendo de una dificultad muy elevada la implementación de una función más compleja o el uso de las predefinidas en bibliotecas, debido a la limitación operacional comentada en el punto [3.3.4] de este proyecto.

4.3 Algoritmos adaptativos

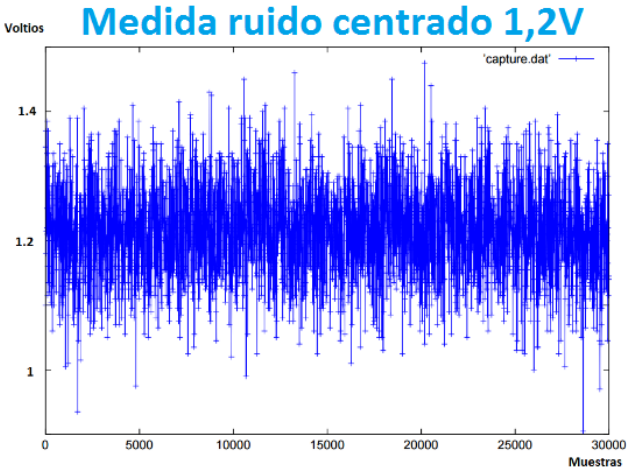
A la vista de esta comparativa, se van a implementar dos algoritmos más. Estos son adaptaciones de la media aritmética y la media armónica. Estos algoritmos evitan que a la hora de mediar las "n" muestras mediadas queden sustituidas por una única muestra resultado. Esto se consigue guardando las muestras en memoria y trabajando con ellas de forma vectorial en vez de directamente. La ventaja de realizar estos algoritmos es la

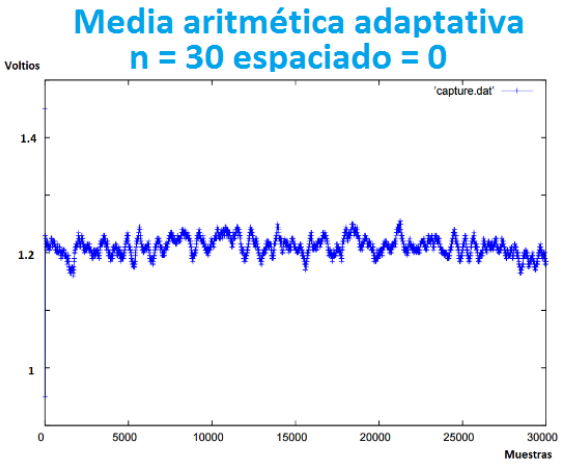
flexibilidad que permiten, pudiendo no sólo variar el número de muestras a mediar, sino el espaciado y las muestras que se pierden al realizar esta operación. Resultan fáciles de adaptar a otros sistemas que trabajen a distintas frecuencias y con distintos requisitos a la hora de realizar la eliminación del ruido generado.

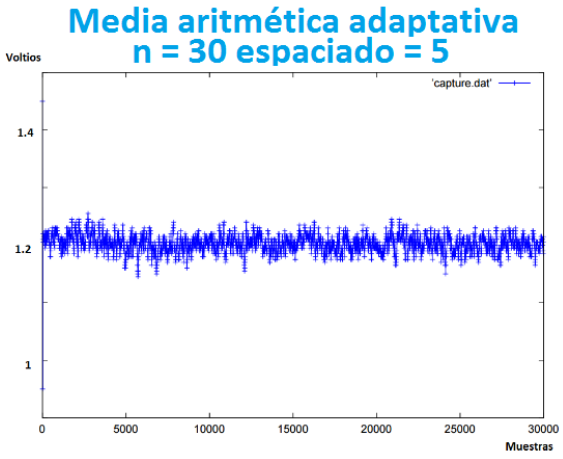


De esta forma obtenemos unos resultados con una mejora enormemente significativa sin apenas aumentar el coste computacional.

- Algoritmo 4 : Media Aritmética Adaptativa







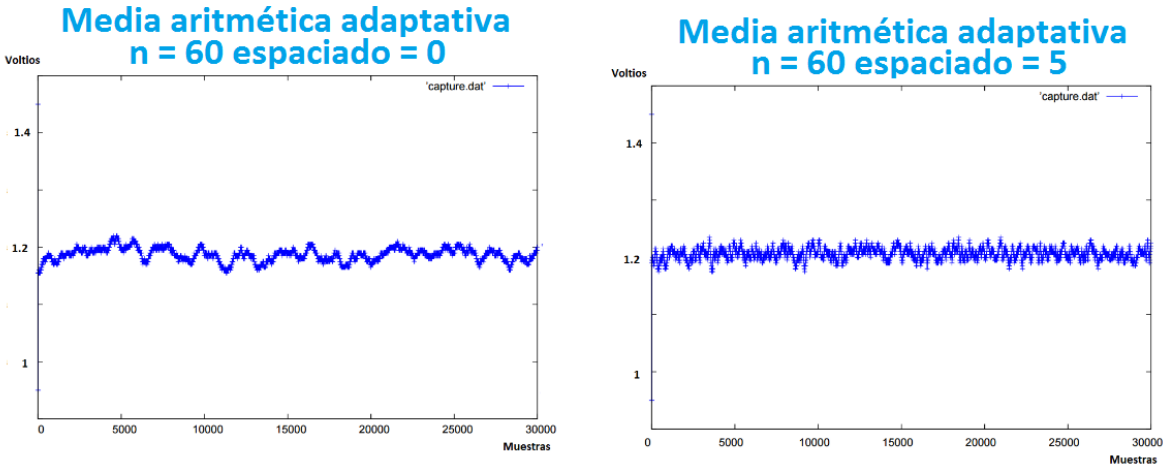
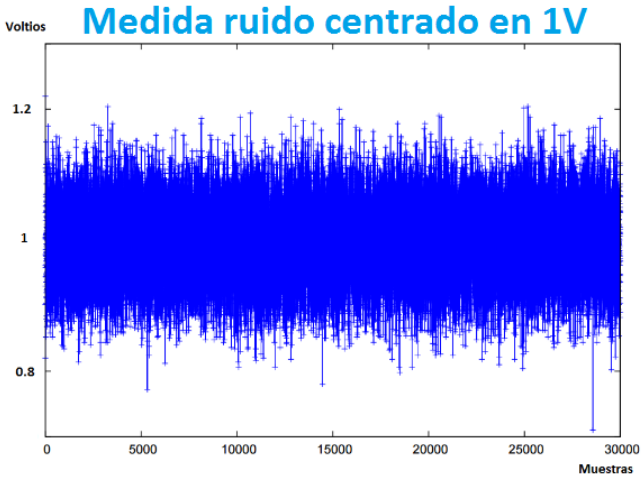
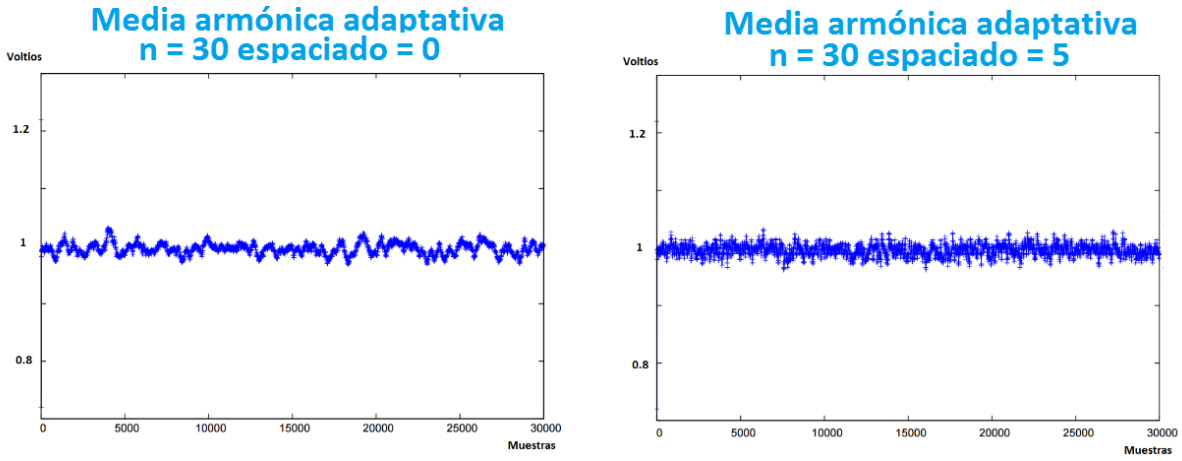


Ilustración 17: Media aritmética adaptativa

- Algoritmo 5 : Media Armónica Adaptativa





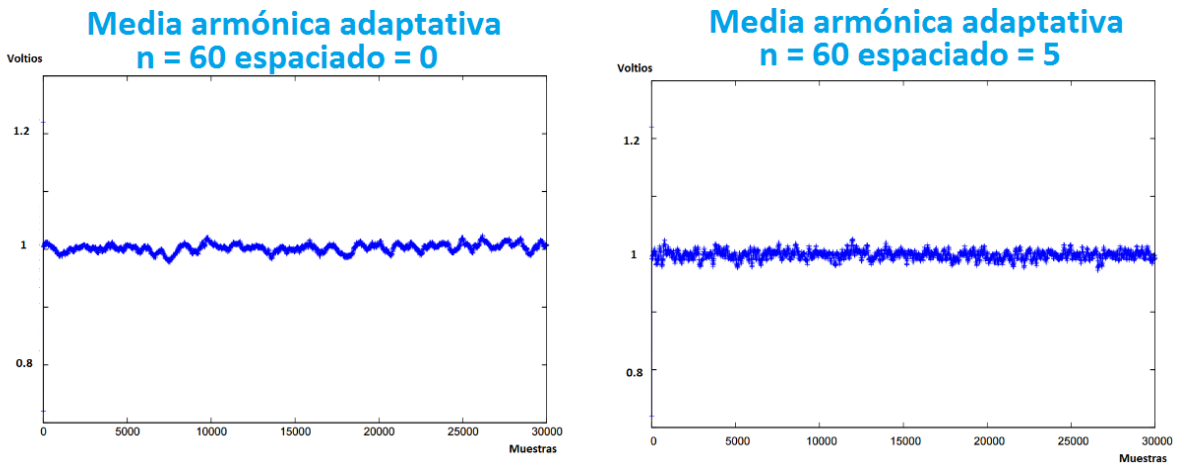


Ilustración 18: Media armónica adaptativa

Analizando los resultados obtenidos, la media aritmética adaptativa ([Ilustración 17]) limpia en mayor medida, el ruido generado, manteniendo la señal íntegra casi en su totalidad, pero en este caso, con menos precisión que la media armónica adaptativa. Este último algoritmo ([Ilustración 18]) destaca sobre los demás por conseguir una amplitud de ruido máxima de 0,02V para un mediado de sesenta muestras frente a la amplitud de ruido máxima de 0,04V de la media aritmética adaptativa, sin embargo, requiere un coste computacional algo mayor.

Los valores en todas las gráficas anteriores están convertidos siguiendo esta fórmula:

$$Voltaje = \frac{(3.3 * muestra\ )}{16000}$$

5 Conclusiones y trabajo futuro

5.1 Conclusiones

En éste Trabajo de fin de Grado se ha llevado a cabo el acoplamiento de un sistema de medida a la BeagleBone Black. Se han desarrollado en lenguaje C cinco algoritmos de reducción de ruido, que se van a aplicar mediante las PRU (procesadores de tiempo real independientes del principal) para la limpieza del posible ruido que pueda generar un sistema de medida.

Analizando los resultados, tras la aplicación de estos algoritmos, observamos que la media aritmética básica destaca sobre el resto de algoritmos básicos implementados (media cuadrática y media armónica) debido posiblemente al bajo coste computacional y la sencillez de las operaciones que realiza, evitando en todo caso el desbordamiento que se produce en los demás. Por otro lado, al implementar los algoritmos adaptativos más complejos, se puede observar que el valor medio más aproximado al real lo consigue la media armónica adaptativa, pese a tener más alto coste computacional que la media aritmética adaptativa.

Adicionalmente se podría reiterar el uso de estos dos algoritmos sobre muestras ya pre-procesadas acercando aún más los valores de la medida al valor real de la misma.

Como conclusión, los dos algoritmos que mejor limpian el ruido, en vista de las pruebas realizadas son la media armónica adaptativa y la media aritmética adaptativa, que a pesar de las limitaciones de compilación a la hora de trabajar con las PRU, la señal resultado de la aplicación de los mismos ha eliminado el ruido casi en su totalidad, pudiendo dar por finalizado este proyecto, habiendo desarrollado un sistema que mejora en gran medida la calidad de la señal muestreada de nuestro sistema de adquisición de datos.

5.2 Trabajo futuro

En vista a los resultados obtenidos, el siguiente paso a realizar podría ser el de crear una aplicación de muestreo a tiempo real que permita ver las muestras una vez aplicados los algoritmos de reducción de ruido. Otra de las posibles modificaciones a realizar sería la del trabajo de forma síncrona de ambas PRU para realizar algoritmos más complejos o poder manejar distintos aparatos de medida al mismo tiempo de una forma mucho más eficiente.

Por último, añadir que otra posible salida ante las limitaciones de las PRU es la utilización de otros sistemas embebidos distintos a las BBB cuyas unidades a tiempo real permitan una mayor facilidad a la hora de implementar algoritmos mucho más complejos con un coste computacional mayor, permitiendo una mayor optimización del sistema completo y una manera mucho más viable de obtener una medida aún más limpia de ruido.

Referencias

[1] Sergio Díaz Bartolomesanz. Universidad Autónoma de Madrid. "Desarrollo de un sistema de medida de parámetros químicos basado en un sistema empotrado" (Junio 2015)

[2] Wikipedia Sistema de adquisición de Datos https://es.wikipedia.org/wiki/Adquisici%C3%B3n_de_datos

[3] Wikipedia Sensor https://es.wikipedia.org/wiki/Sensor

[4] Tranductores. http://es.slideshare.net/oscarx15/catalogo-tipos-de-transductores-y-sensores-extraclase-2do-periodo

[5] Wikipedia. Sistemas Embebidos. https://es.wikipedia.org/wiki/Sistema_embebido

[6] Sistemas Embebidos. https://universodelsaber.wordpress.com/sistema-embebido/

[7] Texas Instruments AM3358. http://www.ti.com/product/AM3358

[8] Datasheet BeagleBone Black: https://www.adafruit.com/datasheets/BBB_SRM.pdf

[9] Ruido en sistemas electrónicos. https://prezi.com/sihux9oeujns/tecnicas-de-reduccion-de-ruido/

[10] Ruido eléctrico. https://sincronizacionymultiplexaje.wordpress.com/2015/10/18/ruido-electrico/

[11] Wikipedia. Media estadística. https://es.wikipedia.org/wiki/Media_(matem%C3%A1ticas)

[12] Datasheet ADS1146: http://www.ti.com/product/ADS1146

[13] Derek Molloy. "Exploring BeagleBone - Tools and Techniques for Building with Embedded Linux"

[14] Manual TI PRU C Compiler: http://processors.wiki.ti.com/index.php/PRU-ICSS

[15] Wikipedia BeagleBoard: https://es.wikipedia.org/wiki/BeagleBoard

[16] J. Godoya, S. Fingerhutha. Técnicas de procesamiento digital de señales para reducción de ruido de señales de múltiples sensores asíncronos.

[17] Aprendizaje automatico : https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico#Algoritmos_gen.C3.A9ticos

Glosario

ADC	Analog to Digital Converter
API	Application Programming Interface
BBB	BeagleBone Black
CS	Chip Select
GPIO	General Purpose Input-Output
HDMI	Hight - Definition Multimedia Interface
MISO	Master Input Slave Output
MOSI	Master Output Slave Input
PRU	Programmable Real Unit
SPI	Serial Peripheral Interface

Anexos

A Comandos de compilación

```
export PRU_SDK=/home/raul/pru_2.0.0B2/
$PRU_SDK/bin/clpru \
--silicon_version=2 \
--hardware_mac=on \
-i$PRU_SDK/include \
-i$PRU_SDK/lib \
-c \
-s \
testHAL.c

$PRU_SDK/bin/clpru \
--silicon_version=2 \
--hardware_mac=on \
-i$PRU_SDK/include \
-i$PRU_SDK/lib \
-O3 \
-c \
-s \
testPRU.c

$PRU_SDK/bin/clpru \
--silicon_version=2 \
--hardware_mac=on \
-i$PRU_SDK/include \
-i$PRU_SDK/lib \
-z \
testPRU.obj testHAL.obj -llibc.a \
-m testPRU.map \
-o testPRU.elf \
AM3359_PRU.cmd

$PRU_SDK/bin/hexpru \
$PRU_SDK/bin.cmd \
./testPRU.elf

echo "Compiling the testBBB.c application"
gcc testBBB.c -o test -lpthread -lprussdrv

echo "Compiling mem2file"
gcc mem2file.c -o mem2file
```

B Código principal

```
#include <stdio.h>
#include <stdlib.h>
#include <prussdrv.h>
#include <pruss_intc_mapping.h>

#define PRU_NUM      0
#define MMAP0_LOC   "/sys/class/uio/uio0/maps/map0/"
#define MMAP1_LOC   "/sys/class/uio/uio0/maps/map1/"

// Short function to load a single unsigned int from a sysfs entry
unsigned int readFileValue(char filename[]){
  FILE* fp;
  unsigned int value = 0;
  fp = fopen(filename, "rt");
  fscanf(fp, "%x", &value);
  fclose(fp);
  return value;
}

int main (void)
{
  int n, ret;
  if(getuid()!=0){
    printf("You must run this program as root. Exiting.\n");
    exit(EXIT_FAILURE);
  }
  /* Initialize structure used by prussdrv_pruintc_intc   */
  /* PRUSS_INTC_INITDATA is found in pruss_intc_mapping.h */
  tpruss_intc_initdata pruss_intc_initdata = PRUSS_INTC_INITDATA;


  unsigned int spiData[3];
  spiData[0] = 0x12FFFF; //INFORMACION PARA EL ADC (comunicacion SPI) para que
comience a medir
  spiData[1] = readFileValue(MMAP1_LOC "addr");
  spiData[2] = readFileValue(MMAP1_LOC "size");
  printf("Sending the SPI Control Data: 0x%x\n", spiData[0]);
  printf("The DDR External Memory pool has location: 0x%x and size: 0x%x bytes\n", spiData[1],
spiData[2]);
  int numberSamples = spiData[2]/2;
  printf("-> this space has capacity to store %d 16-bit samples (max)\n", numberSamples);

  /* Allocate and initialize memory */
  prussdrv_init ();
  ret = prussdrv_open (PRU_EVTOUT_0);
  if(ret){
    printf("Failed to open the PRU-ICSS, have you loaded the overlay?");
    exit(EXIT_FAILURE);
  }

  //prussdrv_pru_write_memory(PRUSS0_PRU1_DATARAM, 0, spiData, 12);  // spi code to PRU1
  prussdrv_pru_write_memory(PRUSS0_PRU1_DATARAM, 0, spiData, 12);
```

```
  /* Map PRU's INTC */
  prussdrv_pruintc_init(&pruss_intc_initdata);

  /* Load the memory data file */
  prussdrv_load_datafile(PRU_NUM, "./data.bin");

  /* Load and execute binary on PRU */
  prussdrv_exec_program (PRU_NUM, "./text.bin");

  /* Wait for event completion from PRU */
  n = prussdrv_pru_wait_event (PRU_EVTOUT_0);  // This assumes the PRU generates an interrupt
                                  // connected to event out 0 immediately before halting
  printf("PRU program completed, event number %d.\n", n);

  /* Disable PRU and close memory mappings */
  prussdrv_pru_disable(PRU_NUM);
  prussdrv_exit ();
  return(EXIT_SUCCESS);
}
```

C Código Funciones

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>

#define EPSILON 0.0000001

void ocp_init()
{
/* enable ocp wide access */
        __asm__ __volatile__
        (
                " LBCO &r0, C4, 4, 4 \n"
                " CLR r0, r0, 4 \n"
                " SBCO &r0, C4, 4, 4 \n"
        );
}

void wait2k(void)
{
        __asm__ __volatile__
        (
                " LDI32   r17, 50000 \n" // TIEMPO DE ESPERA ENTRE MUESTRAS
                "WAIT3: \n"
                " SUB   r17, r17, 1 \n"
                " QBNE  WAIT3, r17, 0 \n"
        );
}

void spiUnselectDevice()
{
        __asm__ __volatile__
        (
                " LDI r17, 70 \n"
                "WAIT: \n"                  // ESPERAMOS ANTES DE PARAR
                " SUB   r17, r17, 1 \n"
                " QBNE  WAIT, r17, 0 \n"
                " SET r30, r30,  5 \n"    // CS a 1
        );
}

void spiEnableSelectDevice()
{
        __asm__ __volatile__
        (
                " CLR r30, r30,  5 \n" // CS a 0
                " LDI r17, 70 \n"
                "WAIT2: \n"                 // ESPERAMOS DESPUES DE INICIAR
                " SUB   r17, r17, 1 \n"
                " QBNE  WAIT2, r17, 0 \n"
        );
}
```

```
uint32_t spiComunication(register uint32_t data, register uint32_t length)
{
// ------------------------------------------------------------
//                  COMUNICACION SPI
// ------------------------------------------------------------
        __asm__ __volatile__
        (
        // Desplazamiento de data para ajustar tamanio
                " LDI r19, 32 \n"
                " SUB r19, r19, r15 \n"
                " LSL r14, r14, r19 \n"
        // -----------------------------------
                " XOR r16, r16, r16 \n" // COLINO
                " ADD r15, r15, 1 \n"
                "SPLICLKMAIN: \n"
                " SUB r15, r15, 1 \n"
                " QBEQ SPIEND, r15, 0 \n"
                " LDI    r17, 70 \n"
                " CLR r30, r30, 2 \n" //COLINO
                "CLKLOW: \n"                    // Parte Baja CLK
                " SUB   r17, r17, 1 \n"
                " QBNE          CLKLOW, r17, 0 \n"
        //----- DEFINE VALOR DEL MOSI -----------
                " QBBC          DATALOW, r14, 31 \n"
                " SET    r30, r30, 1 \n"   // MOSI
                " QBA   DATACONTD \n"
                "DATALOW: \n"
                " CLR   r30, r30, 1 \n"   // MOSI
        //---------------------------------------
                "DATACONTD: \n"
                " LDI r17, 70 \n"
                " SET    r30, r30, 2 \n"   // CLK a 1
                "CLKHIGH: \n"                   // Parte Alta CLK
                " SUB   r17, r17, 1 \n"
                " QBNE          CLKHIGH, r17, 0 \n"
                " LSL    r14, r14, 1 \n"   // desplazamiento DATOS r14
                " CLR   r30, r30, 2 \n"   // CLK a 0
        // ---- OBTIENE VALOR DEL MISO ----------
                " QBBC          DATAINLOW, r31, 3 \n"
                " OR     r16, r16, 0x00000001 \n"
                "DATAINLOW:\n"
                " LSL    r16, r16, 1 \n"
                " JMP SPLICLKMAIN \n"
                "SPIEND: \n"
                " LSR r16, r16, 2 \n"     // elimino bits de mas del SPICLK (2)
                " LDI32 r18, 0x00003FFF \n" // mascara para limpiar
                " AND r16, r16, r18 \n" //le paso la mascara
                " MOV r14, r16 \n"
                " JMP R3.w2 \n"
        );
        /* unreached */
        return 0;
}
```

```
uint32_t sizeOfMemory()
{
// Devuelve el numero de muestras a capturar
	__asm__ __volatile__
	(
		" LDI32 r16, 0x00002000 \n"   // direccion de memoria compartida PRU1
		" LBBO  &r14, r16, 8, 4 \n"     // carga el numero de muestras
		" JMP R3.w2 \n"
	);
	return 0;
}

void writeInMemory(register uint32_t data, register uint32_t offset)
{
// Escribe data en la direccion de memoria incrementada
	__asm__ __volatile__
	(
		" LDI32          r16, 0x00002000 \n"     //direccion de memoria compartida PRU1
		" LBBO&r17, r16, 4, 4 \n"       //carga la direccion que le pasamos en r17
		" ADD r17, r17, r15 \n"
		" SBBO &r14, r17, 0, 2 \n"
	);
}

uint32_t readMemory(register uint32_t offset)
{
// Escribe data en la direccion de memoria incrementada
	__asm__ __volatile__
	(
		" LDI32 r16, 0x00002000 \n"   // direccion de memoria compartida PRU1
		" LBBO  &r17, r16, 4, 4 \n"     // carga la direccion que le pasamos en r17
		" ADD r17, r17, r14 \n"          // añadimos el offset
		" LBBO &r14, r17, 0, 2 \n"       // leemos lo que hay en r17 y lo guardamos en r15
		" JMP R3.w2 \n"
	);
	return 0;
}

float squareroot(float val) {
	float low = 0;
	float high = 10000000; // a sufficiently big number
	float mid = 0;
	while (high - low > EPSILON) {
			mid = low + (high - low) / 2;
			if (mid*mid > val) {
					high = mid;
			} else {
					low = mid;
			}
	}
	return mid;
}
```

D Código medida básica

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>

#include "testHAL.h"

/* the registers for I/O and interrupts */
volatile register unsigned int __R31;
unsigned int i;              // the counter in the time delay

int main()
{
        int muestras;
        float offset = 0;
        int z = 0;
        ocp_init();
    wait2k();
        spiEnableSelectDevice();
    spiComunication(0x02, 8);
    spiUnselectDevice();
        wait2k();
        // Lo configuro para que tome 2000 muestras/seg
        spiEnableSelectDevice();
    spiComunication(0x43010900, 32);
        spiUnselectDevice();
        wait2k();
    spiEnableSelectDevice();
    spiComunication(0x01, 8);
    spiUnselectDevice();
        wait2k();
    spiEnableSelectDevice();
    spiComunication(0x016, 8);
    spiUnselectDevice();
    wait2k();
        //muestras = sizeOfMemory();
        muestras = 100000;
        // while the button r31.0 has not been pressed, keep looping
        while(!(__R31 & 1<<0)){
                spiEnableSelectDevice();
                z = spiComunication(0x12FFFF, 24);
                spiUnselectDevice();
                if (z > 16210) {
                        z = 16210;
                }
                if (z < 0) {
                        z = 0;
                }
                // Escritura en memoria de n muestras
                if (muestras != 0 ){
                        writeInMemory(z, offset);
                }
```

```
            wait2k();
            offset = offset + 2;
            muestras = muestras - 1;

        }
        // Exiting the application - send the interrupt
        __R31 = 35;                 // PRUEVENT_0 on PRU0_R31_VEC_VALID
        __halt();                   // halt the PRU
}
```

E Código media aritmética

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>

#include "testHAL.h"

/* the registers for I/O and interrupts */
volatile register unsigned int __R31;
//unsigned int delay = 25000;
unsigned int i;                 // the counter in the time delay

int main()
{
        int media = 90;
        int muestras;
        float offset = 0;
        int z;
        int suma;

        ocp_init();
        wait2k();
        spiEnableSelectDevice();
        spiComunication(0x02, 8);
        spiUnselectDevice();
        wait2k();
        // Lo configuro para que tome 2000 muestras/seg
        spiEnableSelectDevice();
        spiComunication(0x43010900, 32);
        spiUnselectDevice();
        wait2k();
        spiEnableSelectDevice();
        spiComunication(0x01, 8);
        spiUnselectDevice();
        wait2k();
        spiEnableSelectDevice();
        spiComunication(0x016, 8);
        spiUnselectDevice();
        wait2k();
        //muestras = sizeOfMemory();
        muestras = 30000;
        // while the button r31.0 has not been pressed, keep looping
        while(!(__R31 & 1<<0)){
                // Suma de n muestras
                for (i=0,suma=0; i<media; i++){
                        spiEnableSelectDevice();
                        z = spiComunication(0x12FFFF, 24);
                        spiUnselectDevice();
                        // Limitamos por su valor inferior y superior
                        if (z > 16210) {
                                z = 16210;
                        }
```

```
                        if (z < 0) {
                                z = 0;
                        }
                        suma = suma + z;
                        wait2k();
        }
                // Calculo de la media
                suma = suma / media;

            // Escritura en memoria de n muestras
                if (muestras != 0 ){
                                writeInMemory(suma, offset);
                }

                offset = offset + 2;
                muestras = muestras - 1;

            }
            // Exiting the application - send the interrupt
            __R31 = 35;                 // PRUEVENT_0 on PRU0_R31_VEC_VALID
            __halt();                   // halt the PRU
}
```

F Código media armónica

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>

#include "testHAL.h"

/* the registers for I/O and interrupts */
volatile register unsigned int __R31;
unsigned int i;                 // the counter in the time delay

int main()
{
        int media = 60;
        float muestras;
        int offset = 0;
        float z;
        float suma;
        ocp_init();
        wait2k();
        spiEnableSelectDevice();
        spiComunication(0x02, 8);
        spiUnselectDevice();
        wait2k();
        // Lo configuro para que tome 2000 muestras/seg
        spiEnableSelectDevice();
        spiComunication(0x43010900, 32);
        spiUnselectDevice();
        wait2k();
        spiEnableSelectDevice();
        spiComunication(0x01, 8);
        spiUnselectDevice();
        wait2k();
        spiEnableSelectDevice();
        spiComunication(0x016, 8);
        spiUnselectDevice();
        wait2k();
        //muestras = sizeOfMemory();
        muestras = 30000;
        // while the button r31.0 has not been pressed, keep looping
        while(!(__R31 & 1<<0)){
    // Suma de n muestras
                for (i=0,suma=0; i<media; i++){
                        spiEnableSelectDevice();
                        z = (float)spiComunication(0x12FFFF, 24);
                        spiUnselectDevice();
                        // Limitamos por su valor inferior y superior
                        if (z > 15500) {
                                z = 15500; // Se limita a este valor debido al numero de
decimales que admite
                        }
                        if (z < 0) {
```

```
                                        z = 0;
                        }
                        if (z != 0) {
                                suma = suma + (1/z);
                        }
                        wait2k();
                }
                // Calculo de la media
                if (suma != 0){
                        suma = (float)media / suma;
                }
          // Escritura en memoria de n muestras
                if (muestras != 0 ){
                                writeInMemory(suma, offset);
                }
                offset = offset + 2;
                muestras = muestras - 1;

        }
        // Exiting the application - send the interrupt
        __R31 = 35;                 // PRUEVENT_0 on PRU0_R31_VEC_VALID
        __halt();                   // halt the PRU
}
```

G Código media cuadrática

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include<math.h>

#include "testHAL.h"

/* the registers for I/O and interrupts */
volatile register unsigned int __R31;
unsigned int i;                 // the counter in the time delay

int main()
{
        int media = 30;
        float muestras = 30000;
        int offset = 0;
        int offsetRead = 0;
        float z;
        float suma;

   ocp_init();
        // while the button r31.0 has not been pressed, keep looping
        while(!(__R31 & 1<<0)){
      // Suma de n muestras
      for (i=0,suma=0; i<media ; i++, offsetRead = offsetRead + 2){
                        z = (float)readMemory(offsetRead);
                        z = z * z;
                        suma = suma + z;
      }
              //Calculo de la media
              suma = squareroot(suma);
          // Escritura en memoria de n muestras
          if (muestras != 0 ){
              writeInMemory(z, offset);
          }
              offset = offset + 2;
              muestras = muestras - 1;

        }
        // Exiting the application - send the interrupt
        __R31 = 35;                 // PRUEVENT_0 on PRU0_R31_VEC_VALID
        __halt();                   // halt the PRU
}
```

H Código media aritmética adaptativa

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>

#include "testHAL.h"

/* the registers for I/O and interrupts */
volatile register unsigned int __R31;
unsigned int i;

int main()
{
	int media = 30;
	int muestras = 30000;
	int offset = 0;
	int offsetRead = 0;
	int z;
	int suma;
	float resultado;

	ocp_init();
	//muestras = sizeOfMemory();
	muestras = 10000;
	// while the button r31.0 has not been pressed, keep looping
	while(!(__R31 & 1<<0)){
		// Lectura y mediado de las muestras
		for (i=0,suma=0; i<media ; i++, offsetRead = offsetRead + 2){
			z = (float)readMemory(offsetRead);
			suma = suma + z;
		}

		offsetRead = offsetRead - (media*2) + 2;
		// Reducimos el offsetRead para que vuelva a leer muestras anteriores
		//offsetRead = offsetRead - 58; // Caso sin espacio entre muestras n = 30
		//offsetRead = offsetRead - 48; // Espaciado de 5 muestras n = 30
		//offsetRead = offsetRead - 38; // Espaciado de 10 muestras n = 30
		//offsetRead = offsetRead - 118; // Espaciado entre muestras 0 para n = 60
		//offsetRead = offsetRead - 108; // Espaciado entre muestras 5 para n = 60
		//offsetRead = offsetRead - 88; // Espaciado de 10 muestras 15 para n = 60

		// Calculo de la media
		suma = suma / media;

		// Escritura de las muestras en memoria
		if (muestras != 0 ){
			writeInMemory(suma, offset);
		}
		wait2k();
		offset = offset + 2;
		muestras = muestras - 1;
```

```
        }
        // Exiting the application - send the interrupt
        __R31 = 35;                 // PRUEVENT_0 on PRU0_R31_VEC_VALID
        __halt();                   // halt the PRU
}
```

I Código media armónica adaptativa

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>

#include "testHAL.h"

/* the registers for I/O and interrupts */
volatile register unsigned int __R31;
unsigned int i;

int main()
{
	int media = 60;
	float muestras = 30000;
	int offset = 0;
	int offsetRead = 0;
	float z;
	float suma;

	ocp_init();
	//muestras = sizeOfMemory();
	muestras = 10000;
	// while the button r31.0 has not been pressed, keep looping
	while(!(__R31 & 1<<0)){
		// Lectura y mediado de las muestras
		for (i=0,suma=0; i<media ; i++, offsetRead = offsetRead + 2){
			z = (float)readMemory(offsetRead);
			if (z != 0) {
				suma = suma + (1/z);
			}
		}

		// Reducimos el offsetRead para que vuelva a leer muestras anteriores

		offsetRead = offsetRead - (media*2) + 2; // para cualquier n sin espaciado
		//offsetRead = offsetRead - 48; // Espaciado de 5 muestras n = 30
		//offsetRead = offsetRead - 38; // Espaciado de 10 muestras n = 30
		//offsetRead = offsetRead - 108; // Espaciado entre muestras 5 para n = 60
		//offsetRead = offsetRead - 88; // Espaciado de 10 muestras 15 para n = 60

		// Calculo de la media
		if (suma != 0){
				suma = (float)media / suma;
		}

		// Escritura de las muestras en memoria
		if (muestras != 0 ){
			writeInMemory(suma, offset);
		}
		wait2k();
		offset = offset + 2;
```

```
            muestras = muestras - 1;

        }
        // Exiting the application - send the interrupt
        __R31 = 35;                 // PRUEVENT_0 on PRU0_R31_VEC_VALID
        __halt();                   // halt the PRU
}
```